Give Thor Tools Options

The framework for specifying and using options for Thor Tools is elegant and easy to use.

Tamar E. Granor, Ph.D.

In my last article, I showed how to add your own tools to Thor. This time, I look at how you can provide options for Thor Tools, so that users can customize them.

The VFPX tool, Thor, is a container for developer tools. It comes with dozens of tools, but also allows you to add others. Not surprisingly, when so many tools are available to lots of people, there's disagreement about how some tools should operate.

For example, one of the tools that comes with Thor is Comment Highlighted Text. As the name suggests, it turns whatever lines are currently highlighted into comment lines. It also adds a comment before those lines. By default, that comment says "Removed" and the date. But it's easy to see that different developers might want different versions of that comment.

Thor allows you to set the comment to use. The Options tab of the Thor Configuration form (shown in Figure 1) lets you specify a string to use. (As the figure shows, the string is run through textmerge first.)

So, if you prefer the header comment for removed code to, say, include your initials, you can just add them to the string. **Figure 2** shows the Options tab after I modified the string to include my initials and some additional text.

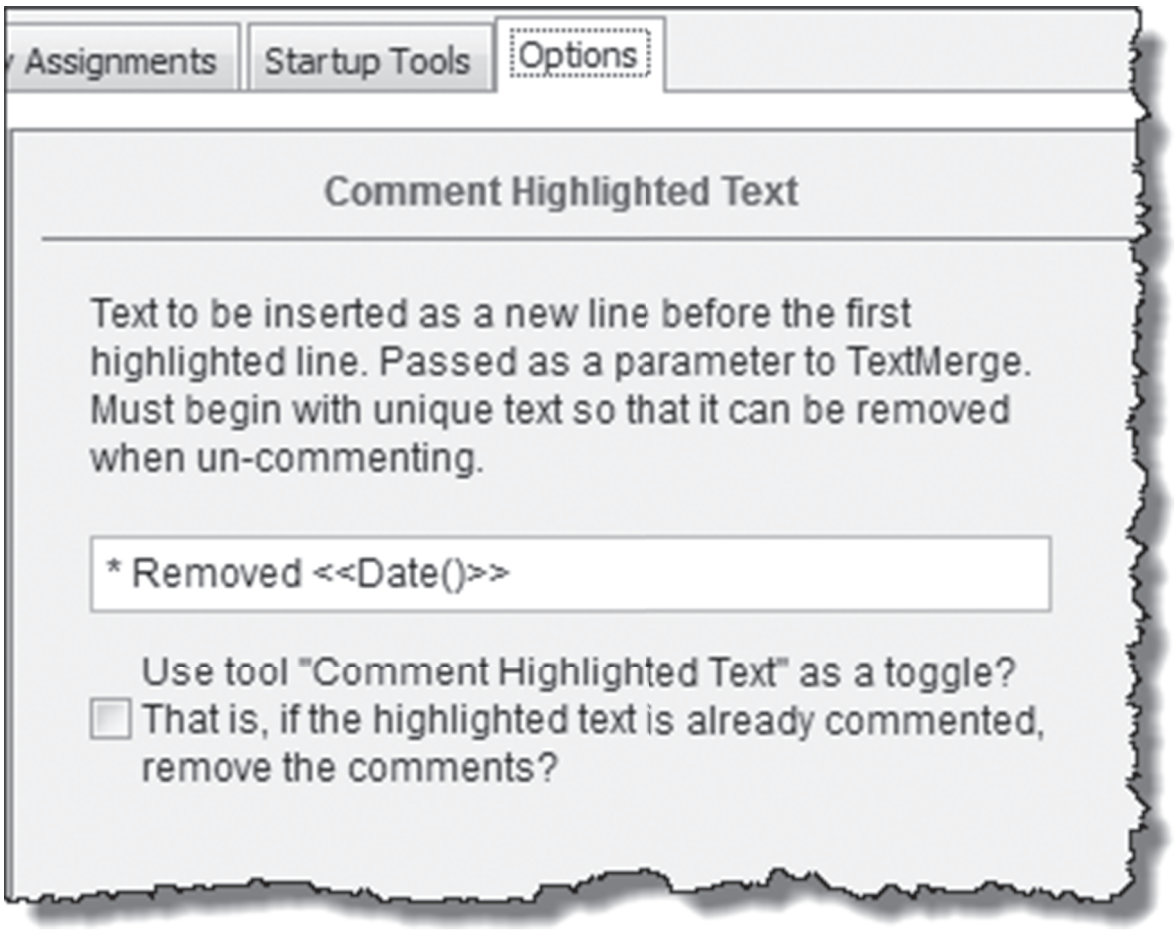

Figure 1. The Options tab of the Thor Configuration dialog lets you specify the string to use as a header comment when you use the Comment Highlighted Text tool.

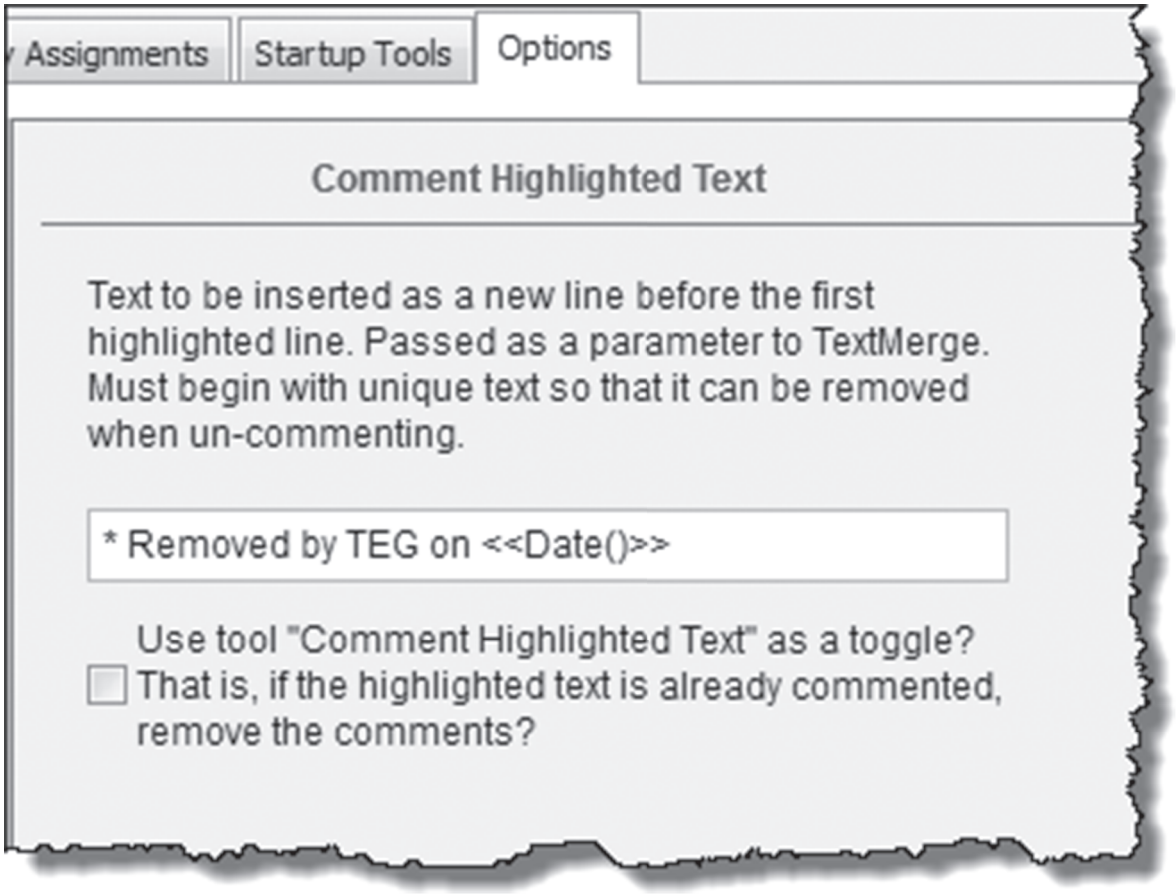

Figure 2. Modifying the header comment on the Options tab changes what the Comment Highlighted Text tool inserts.

The architecture for specifying options for Thor Tools is quite elegant and makes it easy to add options to any tool. There are three elements involved: defining an option, displaying an option and accessing an option.

Viewing and editing Thor tools

Before we look at how to handle each of these elements, let me quickly review how you can see the code for a Thor tool, and how you can customize it.

To open the code for an existing tool, start in either the Launcher or the Configuration form. In either case, highlight the tool you're interested in (in the Configuration form, on the Tool Definitions page) and click the Edit Tool button at the bottom of the right pane.

The Edit File dialog, shown in Figure 3 appears. If you just want to look at the code, click the View this file in Read-Only mode button. If you want to customize the tool, click the Copy this file to folder 'My Tools' and edit it button. In the second case, a copy of the tool code opens; when you save it, it's stored in the My Tools folder of your Thor installation, and from that point on, when you use the tool, the copy in My Tools is used. This allows you to make changes while retaining the original, and means that if the tool is updated in the Thor Repository, when you update Thor, your customized copy remains intact (though, of course, it won't reflect the updates from the Repository).

In most cases, all the elements for specifying options appear in the tool code. You can follow along the examples in the rest of this article by opening the code for the Comment Highlighted Text tool (in read-only mode).

Defining options

The first step in adding options is to tell the tool itself about them. Two parts are needed to do so.

In the top portion of the tool definition, add two properties, OptionClasses and OptionTool. OptionTool indicates what tool should be selected on the Thor Options page when the user clicks the Options button for the tool; having a value assigned to this property also determines whether the Options button appears for the tool. The value in this property is what appears in the left pane of the Options page.

OptionClasses is a comma-separated list of names for classes that define the individual options. Listing 1 shows the two properties as they're specified for the Comment Highlighted Text tool. (As with most of the listings in this article, the code has been slightly reformatted to fit.)

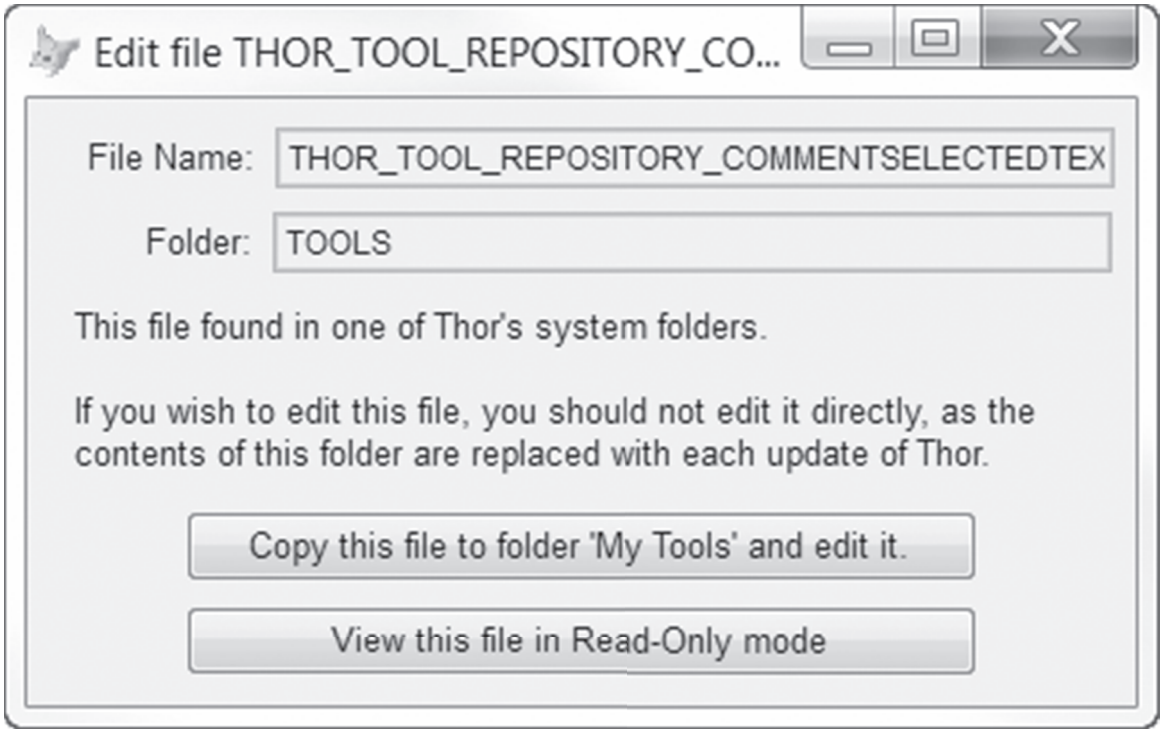

Figure 3. This dialog appears when you click the Edit Tool button in either the Launcher or the Configuration form.

Listing 1. To specify options for a tool, add two propeties to the tool definition.

```
.OptionClasses = 'clsAddComments, ' + ;
                 'clsToggleComments'
.OptionTool    = 'Comment Highlighted Text'
```

Next, you need to define each of the classes specified in the OptionClasses property. You do so by adding them to the tool's PRG file, subclassing each from the Custom class.

Each option class needs four properties, set as follows:

- Tool is the name of the tool, that is, the same value as the OptionTool property.
- Key is a unique name (within the tool) for this option. It's used in looking up the option value.
- Value is the default value for the option. It can be character, numeric, logical or date.
- EditClassName is the name of a container class that contains the instructions for displaying the options for this tool in the Thor Configuration dialog.

The value of EditClassName should be the same for all options for a single tool.

Listing 2 shows the option class definitions for the Comment Highlighted Text tool. The constants they reference appear at the top of the code for the tool and are shown in Listing 3.

Listing 2. Create a custom class to define each option for a tool.

```
Define Class clsAddComments As Custom

  Tool          = ccXToolName
  Key           = ccCommentText
  Value         = '* Removed <<Date()>>'
  EditClassName = ccContainerClassName

Enddefine


Define Class clsToggleComments As Custom

  Tool          = ccXToolName
  Key           = ccToggleComments
  Value         = .F.
  EditClassName = ccContainerClassName

Enddefine
```

Listing 3. The tools that come with Thor use constants to make it easier to manage the tool names and option keys.

```
#Define    ccContainerClassName  ;
           'clsCommentSelectedText'
#Define    ccXToolName           ;
           'Comment Highlighted Text'

#Define    ccCommentText         ;
           'Comment Highlighted Text'
#Define    ccToggleComments      ;
           'Toggle Comments'
```

Displaying options

The next step in providing options is indicating how to display them. All options are shown on the Options tab of the Thor Configuration form (Figure 4.). The left pane shows the list of tools for which options are available. Choose a tool in that list and its options appear in the right pane. Thor and Thor News are always listed first, then other tools in alphabetical order.

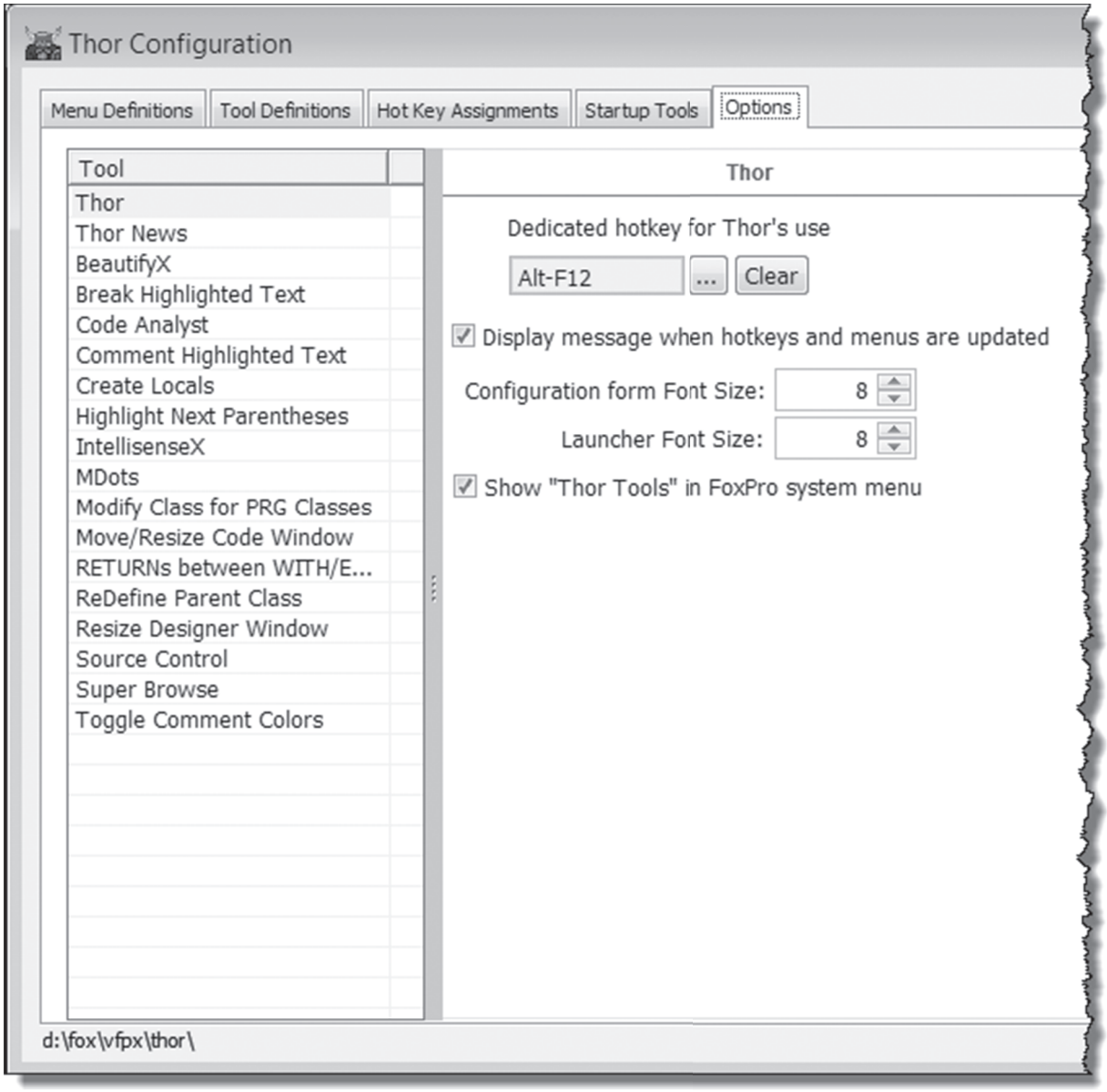

Figure 4. The Options tab of the Thor Configuration form shows the options for each tool that has them. Choose the tool in the left pane to see its options in the right pane.

Options use the VFPX Dynamic Forms project to specify layout. Dynamic Forms lets you specify form layout using a mark-up syntax that looks something like property assignments. A complete explanation of Dynamic Forms is beyond the scope of this article. Fortunately, you can create option pages for tools without knowing too much of it; just model your code after the code used for existing tools.

To specify an options page, you add a definition for the class that was referenced in the EditClassName property of the individual option items. That class needs code only in its Init method. In that code, you instantiate a class called OptionRenderEngine that's built into Thor. Then you set the OptionRenderEngine's cBodyMarkup property to the Dynamic Forms mark-up needed for your options. Finally, you call the OptionRenderEngine's Render method.

Listing 4 shows the clsCommentSelectedText class that's part of the Comment Highlighted Text tool. Note that the values for the Caption properties wrap here, but in the actual tool code, run on a single line.

Listing 4. Displaying options for a Thor Tool relies on the Dynamic Forms project.

```
Define Class clsCommentSelectedText ;
  As Container

Procedure Init
  loRenderEngine = ;
    Execscript(_Screen.cThorDispatcher, ;
               'Class= OptionRenderEngine')

  Text To loRenderEngine.cBodyMarkup ;
    Noshow Textmerge

    .Class     = 'Label'
    .Caption   = 'Text to be inserted as a new
```

```
line before the first highlighted line. Passed
as a parameter to TextMerge. Must begin with
unique text so that it can be removed when un-
commenting.'
    .Width     = 300
    .Left      = 25
    .WordWrap  = .T.
  |
    .Class     = 'TextBox'
    .Width     = 300
    .Left      = 25
    .cTool     = ccXToolName
    .cKey      = ccCommentText
    |
    .Class     = 'CheckBox'
    .Width     = 300
    .Left      = 25
    .WordWrap  = .T.
    .Caption   = 'Use tool "Comment Highlight-
ed Text" as a toggle? That is, if the high-
lighted text is already commented, remove the
comments?'
    .cTool     = ccXToolName
    .cKey      = ccToggleComments

  Endtext

  loRenderEngine.Render(This, ccXToolName)

Endproc
```

There are a few things to note in this code. First, the class is subclassed from the base Container class. All option display classes must be subclassed from Container or a subclass of Container, to provide an object that Thor can simply drop onto the Options page.

You separate specifications for individual controls with the vertical bar ("|"). The code in Listing 4 specifies three controls: a label, a textbox and a checkbox.

For each control, you use actual VFP properties to indicate the layout. Two additional properties are needed for those controls that map to option values. cTool specifies the tool to which the option applies; this is the same name you specify for the OptionTool property in the top portion of the tool definition. cKey is the key for the specific option; this is the same as the Key property of the class that defines the option.

For example, in Listing 4, both the textbox and the checkbox have cTool set to 'Comment Highlighted Text'. The textbox also has cKey set to 'Comment Highlighted Text', which is the key for the option that determines the header comment, while the checkbox has cKey set to 'Toggle Comments', the key specified for the option that determines whether this tool operates as a toggle.

Instantiation of OptionRenderEngine follows the normal Thor style for instantiating classes provided with Thor. It uses a call to ExecScript() passing _Screen.cThorDispatcher as the first parameter and a description of what to do as the second.

Accessing an option

The final part of specifying options is accessing them in tool code, so that their values affect the behavior of the tool. This part turns out to be the easiest of all, because Thor has the mechanism built right in.

To get the value of an option, you use an ExecScript() call in the form shown in Listing 5. You pass the key for the option (as specified in the Key property of the class that defines the option) and the tool name (as specified in the OptionTool property of the tool definition) to indicate which option you want to retrieve.

Listing 5. To retrieve the current value of an option, pass the appropriate parameters to an ExecScript() call to _Screen.cThorDispatcher.

```
uOptionValue = EXECSCRIPT( ;
  _Screen.cThorDispatcher, ;
  "Get Option =", ;
  <Option key>, ;
  <Tool name>)
```

Listing 6 and Listing 7 show code from the Comment Highlighted Text tool that retrieves the two options for that tool and applies them. Note that the Comment Highlighted Text option is character, while the Toggle Comments option is logical. Thor handles the different data types transparently.

Listing 6. This code retrieves the header comment to use for the Comment Highlighted Text tool, and applies textmerge to get the exact string to insert.

```
lcNewLineText = Textmerge(ExecScript( ;
  _Screen.cThorDispatcher, ;
  "Get Option=", ;
  'Comment Highlighted text', ;
  'Comment Highlighted text'))
```

Listing 7. This code retrieves the value of the Toggle Comments option and then applies it to determine whether to comment or uncomment the highlighted text.

```
If ExecScript(_Screen.cThorDispatcher, ;
              "Get Option=", ;
              'Toggle Comments', ;
              'Comment Highlighted text')    ;
   And Left(Ltrim(lcClipText,' ', chr[9]), ;
       Len(lcCommentString)) == ;
         lcCommentString
  loCommentText.RemoveComments(lcClipText, ;
                               lcNewLineText)
Else
  loCommentText.AddComments(lcClipText, ;
                            lcNewLineText)
Endif
```

Invisible options

The Thor options mechanism can also be used to track values behind the scenes, without providing any user interface. The Thor "Get Option=" call has a corresponding "Set Option=" call that allows you to store a value for future reference.

For example, Thor News stores the date it was last displayed, using the code in Listing 8. The code in Listing 9 retrieves the stored date to determine whether it's time to show it.

Listing 8. This code in the Thor News tool stores the date that Thor News was last displayed.

```
ExecScript(_Screen.cThorDispatcher, ;
            "Set Option=", ;
            ccDateLastSeen, ccTool, Date())
```

Listing 9. This code in the Thor News tool retrieves the date Thor News was last displayed.

```
ldDataLastSeen = ExecScript( ;
  _Screen.cThorDispatcher, ;
   "Get Option=", ccDateLastSeen, ccTool)
```

This mechanism means that Thor tools can save information between runs without the tool's author having to come up with a way to do so.

Plenty of examples

One of the great things about Thor is that it comes with lots of sample code, all the tools that come with it. So, if my explanation of Thor options, along with the example of the Comment Highlighted Text tool, isn't sufficient, you can look at almost any Thor tool that has options to see more examples. (Some of the more complex tools use other approaches to showing their options.)

In addition, Thor's model makes the test cycle very short, so you can try adding an option, see whether you got it right and make changes very easily.

Author Profile

Tamar E. Granor, Ph.D. is the owner of Tomorrow's Solutions, LLC. She has developed and enhanced numerous Visual FoxPro applications for businesses and other organizations. Tamar is author or co-author of nearly a dozen books including the award winning Hacker's Guide to Visual FoxPro, Microsoft Office Automation with Visual FoxPro *and* Taming Visual FoxPro's SQL. *Her latest collaboration is* VFPX: Open Source Treasure for the VFP Developer. *Her books are available from Hentzenwerke Publishing (www.hentzenwerke.com). Tamar was a Microsoft Support Most Valuable Professional from the program's inception in 1993 until 2011. She is one of the organizers of the annual Southwest Fox conference. In 2007, Tamar received the Visual FoxPro Community Lifetime Achievement Award. You can reach her at tamar@thegranors.com or through www.tomorrowssolutionsllc.com.*